DWBB Logic – Sequential Overview

The sequential IP consist of high-performance counters, many with either dynamic or static count-to flags. IP in this category have multiple architectures for each function (architecturally optimized for either performance or area) to provide you with the best architecture for your design goals. All IP have a parameterized word length.

For a listing of Building Block components and associated datasheets, see:

- DesignWare Building Block IP User Guide

Basic Counters

The basic counters are binary, Gray code, and Linear Feedback Shift Register (LFSR). They are synchronous with asynchronous reset, and have a synchronous count enable. The binary and LFSR counters are up/down counters, and offer synchronous load.

Count-to Counters

These counters have a count-to architecture that allows the counter to count up or down to a specified value. A terminal count flag is asserted when the terminal value is reached. The terminal count flag can be used to reset the counter. Thus, a simple timer or counter-comparator can be created using this architecture. Three counter structures have this architecture: binary, LFSR, and Gray code.

Additionally, there are two forms of count-to counters: dynamic and static. The count-to value of the dynamic count-to counters can be programmed during the operation of the counter. The static count-to counters have a parameterized count-to value and is, therefore, “hardwired” into the counter. The advantage of the static count-to is that optimization during synthesis, based on constant propagation, eliminates unnecessary logic. Block diagrams for these two structures are shown in Figure 1-1 on page 2.

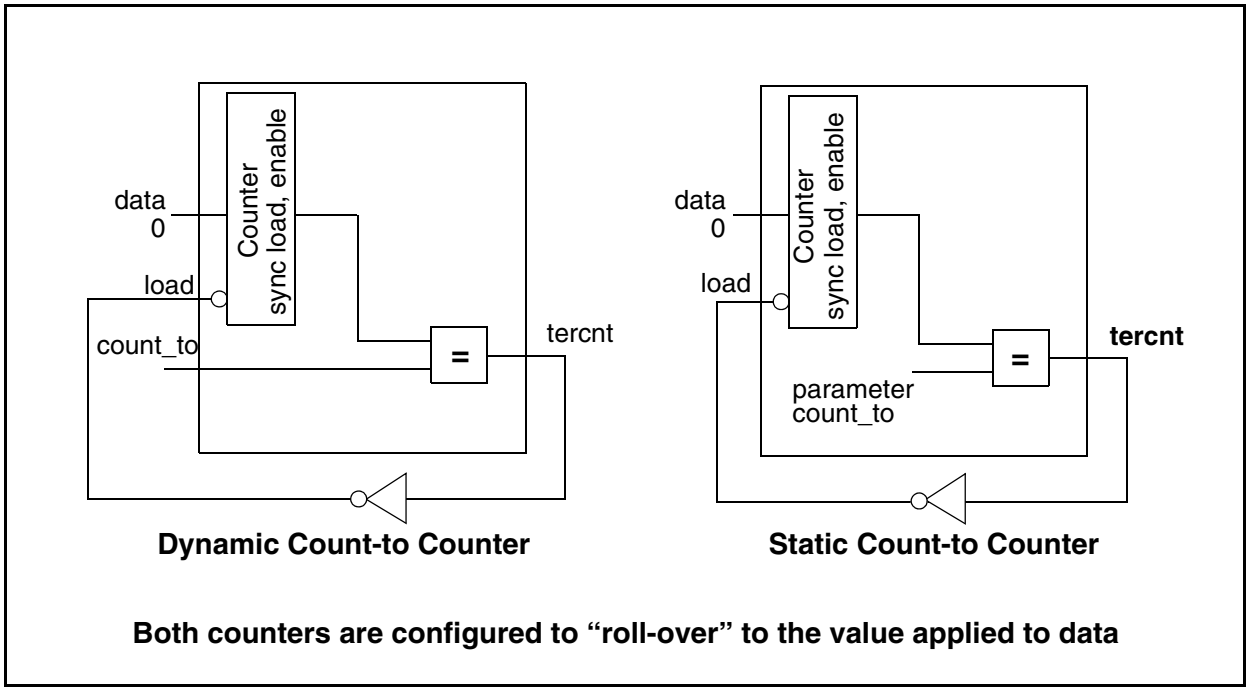
Note that in both diagrams, a simple count-to or timer configuration has been implemented to allow a synchronous load when the counter expires. In a simple configuration, the counter can be configured to count to the count-to value, then synchronously load the new count value. If always counting the same interval, the count value can be a constant, such as zero. An application of this configuration is a programmable timer or event counter counting defined sets of events, such as a pixel and line counter for a video display. In a static count-to counter, the threshold to which we are counting is fixed in hardware (static).

Alternatively, there are applications that require both the count-to threshold and the load count to be variable, as in a dynamic count-to counter. An example application is thresholding or saturating/clipping in filtering applications. In this case, the load data and the count-to threshold are input ports.

LFSR Counter

The Linear Feedback Shift Register has traditionally been used for applications such as pseudo-random number generation. This is due to the fact that the generator sequence is linear but not sequential. An *N*-bit LFSR generates 2^N-1 states while a binary counter has 2^N states. However, the LFSR, which is a shift register with the addition of XORs at defined "taps" within the LFSR, is much faster and smaller than any other counter architecture.

Figure 1-1 Dynamic versus Static Count-to Counters



Because a common counter architecture is the "count-to" or timer type, we have applied this architecture to the LFSR to make it easy to use. The concept is to establish a count-to value that results in a count sequence requiring *N* clocks, where *N* is the "count slice" desired. In a binary counter, for example, you would load *N*-1 into a down counter and count to 0. With the LFSR architecture, you must determine the count-to value necessary to guarantee *N* clocks beginning from the value initially loaded into the counter.

To determine the count-to value, select the polynomial that corresponds to your register width (refer to Table 1-1 on page 3). For example, using a width of 8, the characteristic polynomial is:

$$f(x) = x^8 + x^6 + x^5 + x + 1$$

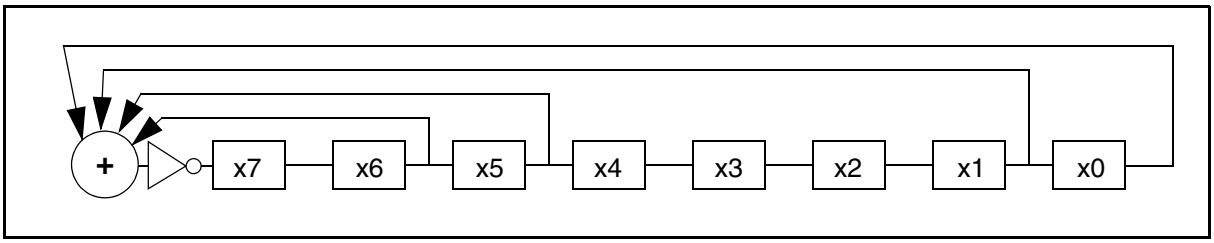
Table 1-1 Primitive Polynomials

Width	Polynomial	Width	Polynomial
1	$x + 1$	26	$x^{26} + x^8 + x^7 + x + 1$
2	$x^2 + x + 1$	27	$x^{27} + x^8 + x^7 + x + 1$
3	$x^3 + x + 1$	28	$x^{28} + x^3 + 1$
4	$x^4 + x + 1$	29	$x^{29} + x^2 + 1$
5	$x^5 + x^2 + 1$	30	$x^{30} + x^{16} + x^{15} + x + 1$
6	$x^6 + x + 1$	31	$x^{31} + x^3 + 1$
7	$x^7 + x + 1$	32	$x^{32} + x^{28} + x^{27} + x + 1$
8	$x^8 + x^6 + x^5 + x + 1$	33	$x^{33} + x^{13} + 1$
9	$x^9 + x^4 + 1$	34	$x^{34} + x^{15} + x^{14} + x + 1$
10	$x^{10} + x^3 + 1$	35	$x^{35} + x^2 + 1$
11	$x^{11} + x^2 + 1$	36	$x^{36} + x^{11} + 1$
12	$x^{12} + x^7 + x^4 + x^3 + 1$	37	$x^{37} + x^{12} + x^{10} + x^2 + 1$
13	$x^{13} + x^4 + x^3 + x + 1$	38	$x^{38} + x^6 + x^5 + x + 1$
14	$x^{14} + x^{12} + x^{11} + x + 1$	39	$x^{39} + x^4 + 1$
15	$x^{15} + x + 1$	40	$x^{40} + x^{21} + x^{19} + x^2 + 1$
16	$x^{16} + x^5 + x^3 + x^2 + 1$	41	$x^{41} + x^3 + 1$
17	$x^{17} + x^3 + 1$	42	$x^{42} + x^{23} + x^{22} + x + 1$
18	$x^{18} + x^7 + 1$	43	$x^{43} + x^6 + x^5 + x + 1$
19	$x^{19} + x^6 + x^5 + x + 1$	44	$x^{44} + x^{27} + x^{26} + x + 1$
20	$x^{20} + x^3 + 1$	45	$x^{45} + x^4 + x^3 + x + 1$
21	$x^{21} + x^2 + 1$	46	$x^{46} + x^{21} + x^{20} + x + 1$
22	$x^{22} + x + 1$	47	$x^{47} + x^5 + 1$
23	$x^{23} + x^5 + 1$	48	$x^{48} + x^{28} + x^{27} + x + 1$
24	$x^{24} + x^4 + x^3 + x + 1$	49	$x^{49} + x^9 + 1$
25	$x^{25} + x^3 + 1$	50	$x^{50} + x^{27} + x^{26} + x + 1$

Next, expand the polynomial generator sequence. This can be done manually or through simulation.

Note that the implementation used by the Foundation LFSR components allows for the zero state, which typical LFSRs prohibit. To achieve this, the module 2 addition of the "taps" is complemented. This structure is illustrated in Figure 1-2 for an LFSR of width 8.

Figure 1-2 Hardware Realization of LFSR (width = 8)



The most significant bit (MSB) of the count sequence for this generator function is:

```
count(7) = not (count(6) xor count(5) xor
count(1) xor count(0))
```

Table 1-2 gives the count sequence for an LFSR of width 8.

Note from this example that if you wanted to create an LFSR counter to count to 9 and then rollover, you would program "10000101" (85H) into the count-to threshold (either by parameter, if static, or applied at the input port `count_to`, if dynamic). This concept can be extended to other count sizes.

Table 1-2 Count Sequence of LFSR (width = 8)

Binary Count	x^7	x^6	x^5	x^4	x^3	x^2	x^1	x^0
0000 (0)	0	0	0	0	0	0	0	0
0001 (1)	1	0	0	0	0	0	0	0
0010 (2)	1	1	0	0	0	0	0	0
0011 (3)	0	1	1	0	0	0	0	0
0100 (4)	1	0	1	1	0	0	0	0
0101 (5)	0	1	0	1	1	0	0	0
0110 (6)	0	0	1	0	1	1	0	0
0111 (7)	0	0	0	1	0	1	1	0
1000 (8)	0	0	0	0	1	0	1	1
1001 (9)	1	0	0	0	0	1	0	1
1010 (A)	0	1	0	0	0	0	1	0

Count-Decode Counters

A count-decode architecture combines a defined type of counter with a corresponding decoder. In the DesignWare Building Block IP, we have provided a binary counter with a binary decoder for implementing a count-select.

Related Topics

- DesignWare Building Block IP User Guide

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com